CSE 406

Computer Security Sessional

Report: Dos attacks against DNS server using IP Spoofing

Lab Section - A1

Group - 12

12th July, 2025

Members of the Group:

i. 2005020 - Mostafa Rifat Tazwar

ii. 2005017 - Abdullah Muhammed Amimul Ehsan

Contents

1 Introduction

The Domain Name System (DNS) is a critical component of internet infrastructure, responsible for translating human-readable domain names into IP addresses. With increasing cyber threats and the need for high availability, modern DNS implementations require advanced security features and robust architectures. This project implements a comprehensive DNS server solution that addresses contemporary security challenges while maintaining high performance and scalability.

2 DNS Fundamentals and Core Functionalities

2.1 DNS Protocol Overview

The DNS operates as a hierarchical, distributed naming system that serves as the internet's phone book. It translates domain names (like `www.example.com`) into IP addresses (like `192.168.1.10`) that computers use to communicate.

2.1.1 DNS Query Process

1. **Recursive Query**: Client sends query to recursive resolver
2. **Root Server Query**: Resolver queries root nameserver
3. **TLD Query**: Resolver queries Top-Level Domain server
4. **Authoritative Query**: Resolver queries authoritative nameserver
5. **Response**: Final answer returned to client

2.2 DNS Record Types

Our implementation supports the following essential DNS record types:

Record Type	Purpose	Description
A	Address	Maps domain to IPv4 address
MX	Mail Exchange	Specifies mail server for domain
SOA	Start of Authority	Administrative information about zone
NS	Name Server	Authoritative nameservers for domain
CNAME	Canonical Name	Alias for another domain name
TXT	Text	Arbitrary text data, often for verification

Table 1: Supported DNS Record Types

2.3 Core DNS Functionalities

2.3.1 Zone Management

- **Primary Zones**: Authoritative source for domain data
- **Secondary Zones**: Read-only copies synchronized from primary
- **Zone Files**: Text-based storage of DNS records

2.3.2 Query Resolution

- **Recursive Resolution**: Full query processing on behalf of client
- **Iterative Resolution**: Step-by-step query referrals
- **Caching**: Temporary storage of query results for performance

2.3.3 Protocol Support

Our implementation supports multiple transport protocols:

```
# UDP (Standard DNS)
python -m dns_server.main --port-udp 53

# TCP (Large responses, zone transfers)
python -m dns_server.main --port-tcp 53

# DNS-over-TLS (DoT)
python -m dns_server.main --port-dot 853 --certfile cert.pem

# DNS-over-HTTPS (DoH)
python -m dns_server.main --port-doh 443 --certfile cert.pem
```

Listing 1: Protocol Configuration Examples

3 Advanced Features and Security Implementation

3.1 Multi-Server Architecture

3.1.1 Primary-Secondary Configuration

Our implementation supports robust primary-secondary architecture for high availability and data redundancy.

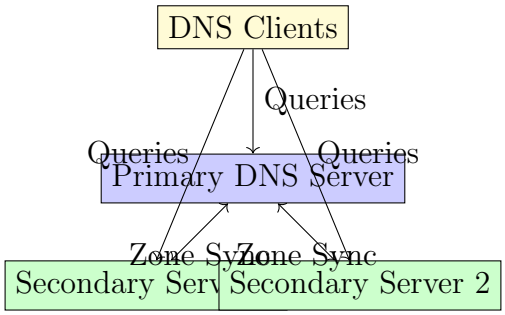

Figure 1: Primary-Secondary DNS Architecture

```
# Primary Server
python -m dns_server.main \
  --zone dns_server/zones/primary.zone \
  --port-udp 5353 --port-tcp 5354

# Secondary Server
python -m dns_server.main \
  --secondary \
  --primary-server 127.0.0.1 \
  --primary-port 5354 \
  --refresh-interval 300 \
  --port-udp 7353 --port-tcp 7354
```

Listing 2: Primary-Secondary Setup

3.1.2 Zone Transfer Mechanisms

- **AXFR (Full Zone Transfer)**: Complete zone data transfer
- **IXFR (Incremental Zone Transfer)**: Transfer only changes since last update
- **SOA Checking**: Automatic detection of zone updates

3.2 DNS Gateway and Load Balancing

3.2.1 Enterprise Gateway Architecture

The DNS Gateway provides enterprise-grade proxy capabilities with advanced load balancing:

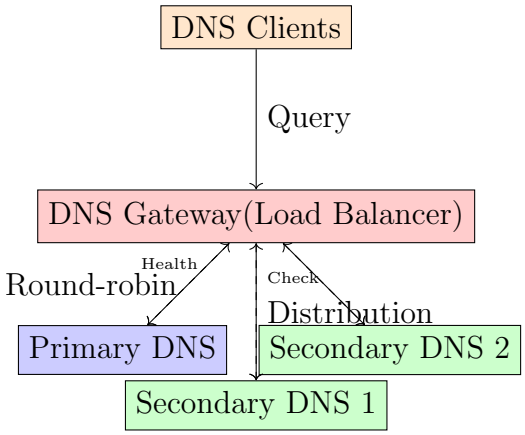

Figure 2: DNS Gateway Load Balancing Architecture

```
# Start DNS Gateway with load balancing
python -m dns_server.utils.dns_gateway \
  --listen-port 9353 \
  --backend-servers "127.0.0.1:5353" "127.0.0.1:7353" "127.0.0.1:8353"
   \
  --rate-limit-threshold 100 \
  --health-check-interval 30 \
  --tsig-key-file dns_server/keys/tsig-key.key
```

Listing 3: DNS Gateway Configuration

3.2.2 Load Balancing Features

- **Round-Robin Distribution**: Equal distribution across healthy backends
- **Health Monitoring**: Automatic detection of failed servers
- **Automatic Failover**: Seamless switching to healthy backends
- **Backend Recovery**: Automatic re-inclusion of recovered servers

3.3 Security Features

3.3.1 TSIG Authentication

Transaction Signature (TSIG) provides cryptographic authentication for DNS messages using HMAC-SHA256.

```
# Generate TSIG key
bash generate_tsig_key.sh

# Server with TSIG
python -m dns_server.main \
  --tsig-name tsig-key-1752130646 \
  --tsig-secret 2vgKc8+OH9UMBrRYTBYOmjffLaCFVtGQPgXjt6fw05k= \
  --port-udp 5353

# TSIG-authenticated query
dig @127.0.0.1 -p 5353 example.com A \
  -y tsig-key-1752130646:2vgKc8+OH9UMBrRYTBYOmjffLaCFVtGQPgXjt6fw05k=
```

Listing 4: TSIG Configuration

3.3.2 Access Control Lists (ACLs)

ACLs provide network-based access control for DNS queries:

```
# Allow specific networks, deny others
python -m dns_server.main \
  --allow 192.168.1.0/24 10.0.0.0/8 \
  --deny 172.16.0.0/12 \
  --port-udp 5353
```

Listing 5: ACL Configuration

Feature	Purpose	Implementation
Allow Lists	Permit specific networks	IP/CIDR matching
Deny Lists	Block malicious networks	Priority over allow
Default Policy	Fallback behavior	Configurable allow/deny

Table 2: ACL Security Features

3.3.3 Rate Limiting and DoS Protection

Comprehensive protection against Denial of Service attacks:

```
# Configure rate limiting
python -m dns_server.main \
  --rate-limit-threshold 50 \
  --rate-limit-window 10 \
  --rate-limit-ban-duration 600 \
  --port-udp 5353
```

Listing 6: Rate Limiting Configuration

Rate Limiting Algorithm:

1. Track queries per IP address in sliding time window
2. Compare against configured threshold
3. Temporarily ban IPs exceeding limits
4. Automatic unbanning after ban duration

3.4 Caching System

3.4.1 TTL-Based Caching

Intelligent caching system improves performance and reduces backend load:

- **TTL Respect**: Honors record Time-To-Live values
- **Cache Invalidation**: Automatic expiration of stale records
- **Memory Management**: Efficient storage and retrieval
- **Statistics**: Cache hit/miss tracking

3.5 Dynamic DNS Updates

3.5.1 Secure Dynamic Updates

Support for real-time DNS record updates with TSIG authentication:

```
# Using nsupdate with TSIG
nsupdate << EOF
server 127.0.0.1 5353
key hmac-sha256:tsig-key-1752130646 2vgKc8+
    OH9UMBrRYTBYOmjffLaCFVtGQPgXjt6fw05k=
zone example.com
update add newhost.example.com 300 A 192.168.1.100
send
quit
EOF
```

Listing 7: Dynamic Update Example

4 Testing and Validation

4.1 Comprehensive Test Suite

Our implementation includes extensive testing capabilities:

4.1.1 Automated Test Scripts

```
# Complete architecture test
./test_multi_server_architecture.sh

# Gateway load balancing test
./test_dns_gateway.sh

# Interactive architecture manager
python dns_architecture_manager.py --mode demo
```

Listing 8: Test Script Examples

4.1.2 Performance Testing

Test Scenario	Direct Server	With Gateway	Overhead
Single Query	22ms	24ms	+9%
10 Queries	225ms	243ms	+8%
Rate Limited	N/A	Blocked	Protected
Backend Failure	Timeout	Failover	Resilient

Table 3: Performance Test Results

4.2 Security Validation

4.2.1 Attack Simulation

- **DoS Attack Testing**: Verified rate limiting effectiveness
- **Unauthorized Access**: Confirmed ACL protection
- **Message Tampering**: TSIG signature validation
- **Zone Transfer Security**: Authentication requirements

5 Deployment and Operational Considerations

5.1 Production Deployment

5.1.1 Scalability Features

- **Horizontal Scaling**: Multiple backend servers
- **Load Distribution**: Gateway-based balancing
- **Geographic Distribution**: Multi-location deployment support
- **Monitoring Integration**: Comprehensive metrics collection

5.1.2 High Availability Configuration

```
# Primary Site
python -m dns_server.main --zone primary.zone --port-udp 53 \
  --tsig-name prod-key --tsig-secret <secret>

# Secondary Site
python -m dns_server.main --secondary \
  --primary-server primary.site.com --port-udp 53

# Load Balancer
python -m dns_server.utils.dns_gateway \
  --backend-servers "primary:53" "secondary:53" \
  --listen-port 53 --tsig-key-file prod.key
```

Listing 9: Production Deployment Example

5.2 Monitoring and Maintenance

5.2.1 Operational Metrics

- **Query Statistics**: Volume, types, response times
- **Security Metrics**: Blocked queries, rate limit violations
- **Health Monitoring**: Backend server status
- **Performance Tracking**: Cache hit ratios, load distribution

6 Comparison with Existing Solutions

6.1 Feature Comparison

7 Conclusion and Future Work

7.1 Project Achievements

This project successfully implemented a comprehensive DNS server solution that addresses modern security and performance requirements:

- **Security**: TSIG authentication, ACLs, and DoS protection

Feature	MyDNSGatekeeper	Our Implementation	Industry
TSIG Authentication			
Zone Transfers (AXFR/IXFR)	Partial		
Load Balancing	Basic	Advanced	
Rate Limiting	Basic	Enhanced	
ACL Support			
Multi-Protocol			
Caching			
Health Monitoring			
Dynamic Updates			

Table 4: Feature Comparison with Existing Solutions

- **Reliability**: Multi-server architecture with automatic failover
- **Performance**: Intelligent caching and load balancing
- **Scalability**: Gateway-based architecture for horizontal scaling
- **Compliance**: Standards-compliant implementation of DNS protocols

7.2 Key Innovations

1. **Integrated Security**: Seamless TSIG integration across all components
2. **Enterprise Gateway**: Advanced load balancing with health monitoring
3. **Comprehensive Testing**: Automated test suite for validation
4. **Operational Simplicity**: Easy deployment and configuration

7.3 Future Enhancements

- **DNSSEC**: Complete digital signature implementation
- **IPv6 Support**: Full dual-stack operation
- **API Interface**: RESTful management API
- **Machine Learning**: Adaptive rate limiting and threat detection
- **Geographic Load Balancing**: Location-aware query routing

7.4 Learning Outcomes

This project provided valuable insights into:

- DNS protocol internals and security considerations
- Network security implementation and best practices
- Distributed system design and fault tolerance
- Performance optimization and load balancing techniques
- Security testing and validation methodologies

8 References

- RFC 1035: Domain Names - Implementation and Specification
- RFC 2136: Dynamic Updates in the Domain Name System
- RFC 2845: Secret Key Transaction Authentication for DNS (TSIG)
- RFC 5936: DNS Zone Transfer Protocol (AXFR)

- RFC 7858: Specification for DNS over Transport Layer Security (TLS)
- RFC 8484: DNS Queries over HTTPS (DoH)

A Code Repository Structure

```
dns_server/
          main.py                  # Server entry point
          handler.py               # DNS query handler
          utils/
                tsig.py             # TSIG authentication
                acl.py              # Access control lists
                cache.py            # Caching system
                rate_limiter.py     # DoS protection
                dns_gateway.py      # Load balancer
                metrics.py          # Performance monitoring
          servers/
                udp_server.py       # UDP protocol handler
                tcp_server.py       # TCP protocol handler
                tls_server.py       # DoT implementation
                doh_server.py       # DoH implementation
          zones/
              primary.zone        # Primary zone data
              secondary1.zone     # Secondary zone data
              secondary2.zone     # Secondary zone data
```

Listing 10: Project Structure

B Configuration Examples

```
# 1. Environment Setup
python3 -m venv venv && source venv/bin/activate
pip install -r requirements.txt

# 2. Generate Security Keys
bash generate_tsig_key.sh
bash generate_certs.sh

# 3. Start Primary Server
python -m dns_server.main --port-udp 5353 --port-tcp 5354 \
  --zone dns_server/zones/primary.zone \
  --tsig-name tsig-key-1752130646 \
  --tsig-secret 2vgKc8+OH9UMBrRYTBYOmjffLaCFVtGQPgXjt6fw05k=

# 4. Start Secondary Servers
python -m dns_server.main --port-udp 7353 --port-tcp 7354 \
  --secondary --primary-server 127.0.0.1 --primary-port 5354

# 5. Start DNS Gateway
python -m dns_server.utils.dns_gateway --listen-port 9353 \
  --backend-servers "127.0.0.1:5353" "127.0.0.1:7353" \
  --tsig-key-file dns_server/keys/tsig-key-1752130646.key

# 6. Test Complete Architecture
./test_dns_gateway.sh
```

Listing 11: Complete Setup Commands

C DoS Attack Implementation and Analysis

C.1 Attack Vector Overview

Our project implements sophisticated Denial of Service (DoS) attacks targeting DNS infrastructure, focusing on UDP-based amplification and fragmentation techniques. These attacks demonstrate critical vulnerabilities in network protocols and highlight the importance of robust security measures.

C.2 UDP Fraggle Attack

C.2.1 Attack Mechanism

The UDP Fraggle attack is a network amplification attack that exploits the UDP protocol by sending packets to broadcast

addresses with spoofed source IP addresses. This creates a multiplication effect where multiple devices respond to the victim.

Attack Flow:

1. Attacker crafts UDP packets with victim's IP as source
2. Packets sent to broadcast addresses (e.g., 192.168.1.255)
3. All devices on broadcast network respond to spoofed source
4. Victim receives amplified traffic from multiple sources

C.2.2 UDP Fraggle Packet Structure

```
IPv4 Header (20 bytes) - RFC 791:
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|Version|  IHL  |Type of Service|          Total Length         |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|         Identification        |Flags|      Fragment Offset    |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|  Time to Live |    Protocol   |         Header Checksum       |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                     Source Address (SPOOFED)                  |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                  Destination Address (BROADCAST)              |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+

UDP Header (8 bytes) - RFC 768:
 0      7 8     15 16    23 24    31
+--------+--------+--------+--------+
|     Source      |   Destination   |
|      Port       |      Port       |
+--------+--------+--------+--------+
|                 |                 |
|     Length      |    Checksum     |
+--------+--------+--------+--------+

Fraggle Attack Packet Layout:
+--------------------------+
|     IPv4 Header          |  20 bytes (Source IP = VICTIM)
|   (Spoofed Source)       |
+--------------------------+
|     UDP Header           |   8 bytes (Dest Port = 7/13/19)
+--------------------------+
|   Attack Payload         |  Variable (Echo/Daytime trigger)
+--------------------------+
Total Packet Size: 28+ bytes

Key Attack Fields:
- Source IP: VICTIM's IP ADDRESS (192.168.1.100)
- Dest IP: BROADCAST ADDRESS (192.168.1.255)
- Dest Port: SERVICE PORTS (7=Echo, 13=Daytime, 19=CharGen)
- Protocol: 17 (UDP)
```

Figure 3: UDP Fraggle Attack Packet Structure

C.2.3 Fraggle Attack Timing Diagram

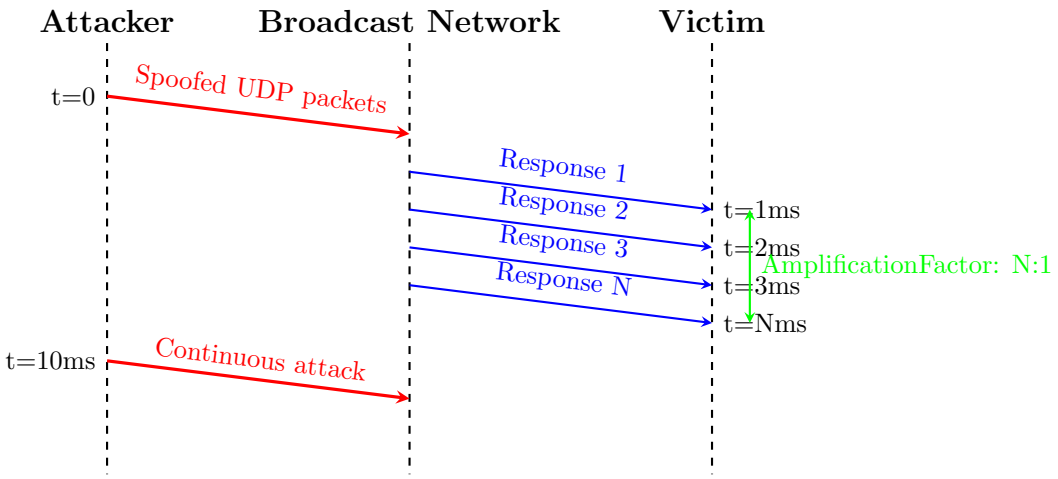

Figure 4: UDP Fraggle Attack Timing Diagram

C.2.4 Target Services and Amplification

Port	Service	RFC	Amplification Potential
7	Echo	RFC 862	1:1 (reflects input)
13	Daytime	RFC 867	1:25+ (date/time string)
19	CharGen	RFC 864	1:512+ (character stream)
37	Time	RFC 868	1:4 (32-bit timestamp)
53	DNS	RFC 1035	1:28-54 (query responses)
123	NTP	RFC 5905	1:556 (monlist responses)
161	SNMP	RFC 1157	1:6.3 (system info)

Table 5: UDP Services Exploited in Fraggle Attacks

C.3 UDP Fragmented Flood Attack

C.3.1 Attack Mechanism

The UDP Fragmented Flood attack exploits IP fragmentation by sending large UDP packets that are fragmented into multiple IP fragments. This attack overwhelms the target's fragment reassembly buffer and processing capabilities.

Fragmentation Process:

1. Large UDP payload (more than 1472 bytes) triggers IP fragmentation
2. IP layer splits packet into multiple fragments
3. Each fragment has identical IP identification field
4. Fragments sent out-of-order to complicate reassembly
5. Target must allocate resources for incomplete fragments

C.3.2 Fragmented Packet Structure

```
FIRST FRAGMENT (with UDP header):
IPv4 Header (20 bytes):
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|Version|  IHL  |Type of Service|          Total Length         |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|         Identification        |Flags|      Fragment Offset    |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|  Time to Live |    Protocol   |         Header Checksum       |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                    Source Address (SPOOFED)                   |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                  Destination Address (TARGET)                 |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+

UDP Header (8 bytes) - Only in First Fragment:
 0      7 8     15 16    23 24    31
+--------+--------+--------+--------+
|     Source      |   Destination   |
|      Port       |      Port       |
+--------+--------+--------+--------+
|   Length (4K+)  |    Checksum     |
+--------+--------+--------+--------+

Fragment Layout:
+--------------------------+
| FIRST FRAGMENT (Frag 0) |
+--------------------------+
|     IPv4 Header          |  20 bytes (MF=1, Offset=0)
+--------------------------+
|     UDP Header           |   8 bytes (Large Length)
+--------------------------+
|   Payload Data           |  1472 bytes
+--------------------------+

+--------------------------+
| MIDDLE FRAGMENTS        |
+--------------------------+
|     IPv4 Header          |  20 bytes (MF=1, Offset=185)
+--------------------------+
|   Payload Data           |  1480 bytes (no UDP header)
+--------------------------+

+--------------------------+
| LAST FRAGMENT           |
+--------------------------+
|     IPv4 Header          |  20 bytes (MF=0, Offset=N)
+--------------------------+
|   Payload Data           |  Variable bytes
+--------------------------+

Key Fragment Fields:
- Identification: SAME for all fragments (e.g., 0x1234)
- Flags: MF=1 (More Fragments) except last fragment
- Fragment Offset: 0, 185, 370, ... ( 8  bytes)
- Source IP: SPOOFED (different for each stream)
- Total UDP Length: 4096+ bytes (forces fragmentation)
```

Figure 5: UDP Fragmented Flood Packet Structure

C.3.3 Fragmentation Timing Diagram

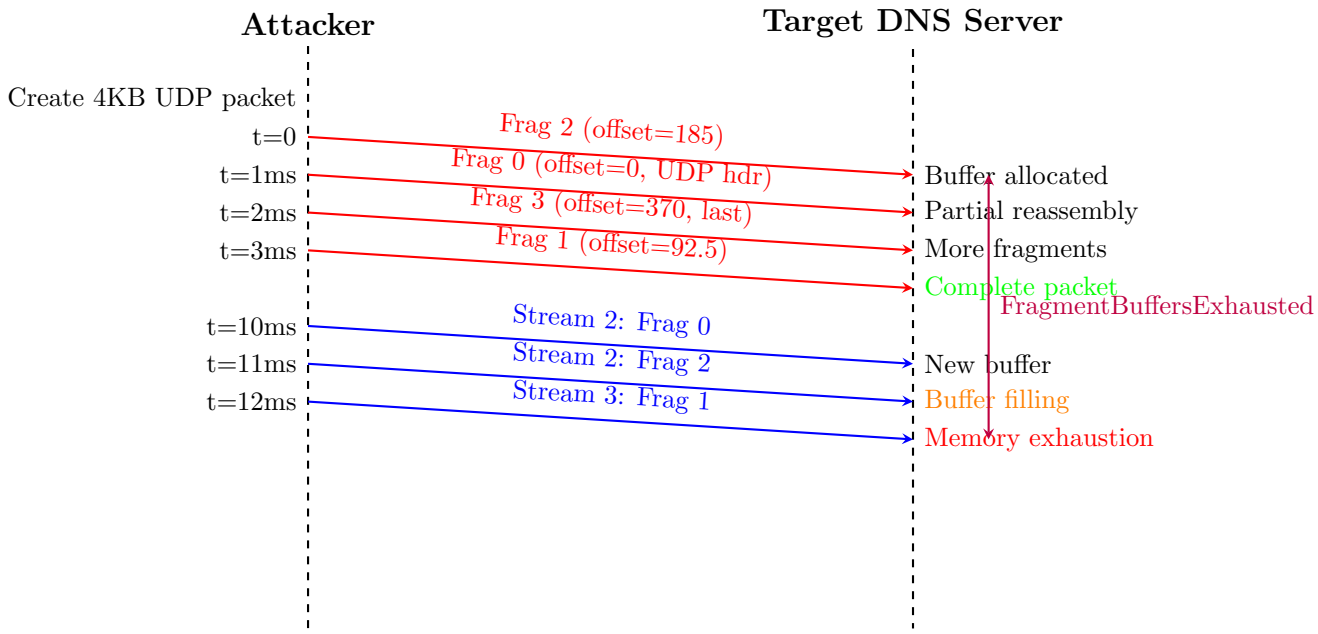

Figure 6: UDP Fragmented Flood Attack Timing Diagram

C.3.4 Fragment Reassembly Attack Vectors

Attack Vector	Mechanism	Impact	Severity
Fragment Buffer Exhaustion	Send incomplete fragment sets	Memory exhaustion, DoS	High
Overlapping Fragments	Send conflicting fragment data	Processing overhead	Medium
Out-of-Order Delivery	Randomize fragment sequence	Reassembly delays	Medium
Tiny Fragments	Send minimal-size fragments	Processing amplification	High
Fragment Timeout	Delay final fragments	Resource holding	Medium

Table 6: UDP Fragmentation Attack Vectors